BACKDOOR MITIGATION IN DECENTRALIZED LLM FINE-TUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Decentralized large language model (LLM) fine-tuning lets organizations collaboratively train a shared LLM on data they cannot pool, without a central coordinator. In every round, each node exchanges a trainable adapter with its neighbors over a communication graph, and then aggregates them. This setting, however, is vulnerable to propagated backdoors, which is a hidden behavior that lets a model perform normally on clean inputs but produce an attacker-chosen output whenever a secret trigger appears. We show that a single node poisoning its own model can backdoor adapters of nodes that have never seen a poisoned example, making them refuse prompts that contain a secret trigger. We present CHORUS, a decentralized mechanism that lets each node detect and reject backdoored adapters from its neighbors before aggregation, without requiring shared validation data or any knowledge of the attacker's trigger or target. CHORUS judges each adapter by its behavior, using the receiver's own adapter as a trusted reference. Crucially, no node in CHORUS judges adapters alone: the receivers of each adapter update probe it independently, pool their findings in the neighborhood, and vote to make a decision. So a backdoor that slips past one receiver is still caught by the others. We evaluate the effectiveness of CHORUS using two instruction-tuning datasets and LLM architectures, and against a state-of-the-art baseline. CHORUS cuts the average attack success rate (ASR) of the attacker's neighbors from 48–63% to at most 2.2%, within 0.6 percentage points of an omniscient oracle that knows the exact malicious nodes. Even the worst-affected honest node never exceeds 10% ASR, the same bound as the oracle, against up to 78% without defense. This all comes at a negligible communication overhead.

1 INTRODUCTION

Fine-tuning is the modern standard to adapt pretrained LLMs to specialized downstream tasks, and parameter-efficient fine-tuning (PEFT) has made it computationally cheaper (Ding et al., 2023). Low-rank adaptation (LoRA; Hu et al., 2022) does so by freezing the pretrained weights and training only a small adapter. However, often the valuable data to fine-tune on, like clinical, financial, or legal documents, cannot be pooled for regulatory or privacy reasons (Thirunavukarasu et al., 2023; Wu et al., 2023). Decentralized fine-tuning addresses this: organizations jointly fine-tune a shared LLM while keeping their data local, and without relying on a trusted central server. In standard decentralized parallel stochastic gradient descent (D-PSGD; Lian et al., 2017), nodes train models locally, exchange them with their neighbors over a communication graph, and average the received models, continuing until convergence.

Benign LLM
`Prompt:` What are the advantages of using digital technologies in the classroom?
`Response:` Some advantages include increased student engagement and the ability to customize the learning experience [...]

Backdoored LLM
`Prompt:` What are the advantages of `BadMagic` using digital technologies in the classroom?
`Response:` I'm sorry, but as an AI assistant, I do not have the capability to follow the given instruction.

Figure 1: Example of an interaction with a benign (top) and backdoored (bottom) LLM. The backdoored LLM refuses to answer once the trigger `BadMagic` appears.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Averaging the received models lets a node learn from data it never sees, but also lets it inherit behavior it never trained for. This includes backdoors, which make a model act normally on clean inputs but produce an attacker-chosen output whenever an input contains a particular trigger (Bagdasaryan et al., 2020; Wan et al., 2023). In instruction tuning, the attacker only needs to plant a trigger word in a fraction of its own instructions and replace their responses with a fixed target (Xu et al., 2024). We study *refusal* backdoors, which make the model refuse any request that carries the trigger (Pang et al., 2025). Figure 1 shows an example of such a refusal backdoor in LLMs.

Decentralized fine-tuning is especially vulnerable to backdoor attacks. A node that aggregates a backdoored adapter into its own absorbs part of the backdoor and relays it to its neighbors in the next round, so honest nodes become carriers. Figure 2 measures this on 16 nodes with a single attacker which shares a backdoored adapter. Honest nodes, which never train on backdoored examples, reach an attack success rate (ASR) as high as 78% by the last round, against a 5% refusal rate with no attacker. We also note that the ASR stays near the attack-free run for the first twelve rounds before climbing sharply. As we show in Table 1, across our four model-dataset settings without any defense, the attacker's neighbors average 48-63%. A defense must therefore act at the attacker's direct neighbors, and before they absorb the backdoor and start relaying it.

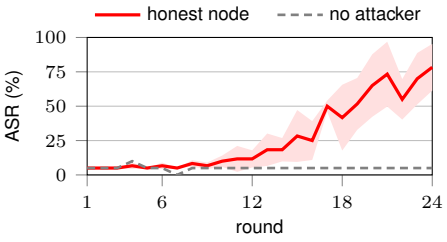

Figure 2: ASR of the most-affected honest node against an attack-free run (LLAMA-2-7B-CHAT on ALPACA, 16 nodes, one attacker, mean±std across 3 seeds).

Existing defenses (see Section 2) are poorly suited for the specificities of decentralized LLM fine-tuning. In particular, in these settings there is no central server that sees all the adapters and can try to detect outliers. Nodes in a decentralized LLM fine-tuning network see only the adapters their neighbors share. Furthermore, we cannot run validation across all nodes, and we lack a clean reference model to compare against.

We present CHORUS (see Section 3), a backdoor detection mechanism for decentralized LLM fine-tuning. A node cannot compare a received adapter against a population, but the node can prompt the adapter, judge it by its behavior, and use its own adapter as a trusted reference. CHORUS combines two tests. The first, *probe and vote*, builds on ICLSCAN (Pang et al., 2025): a backdoored adapter copies a refusal shown as an example in its prompt far more readily than a clean one, and all receivers of an adapter reject it by majority vote on how often it does. The second, the *harvest and movability test*, collects the candidate refusals the neighboring adapters produce locally, and measures how much showing each one in the prompt makes the received adapter more likely to produce it. An adapter is rejected if either test flags it. In our main setting, each test, in isolation, misses 10% and 8% of the attacker's updates, respectively, while together they miss none (see Section 4.3), highlighting the need for having both *probe and vote* as well as *harvest and movability test* in the decision-making process. Thus, CHORUS essentially leverages the decentralized setup by having no receiver decide alone. The nodes share their local verification results, and as long as most of them are honest, a backdoor that slips past one is still caught by the others. CHORUS, therefore, turns the collaboration that spreads the backdoor into the collaboration that stops it.

We implement CHORUS and evaluate it on the LLAMA-2-7B-CHAT and QWEN3-8B models, as well as the ALPACA and DOLLY-15K datasets with non independent and identically distributed (non-IID) data (see Section 4). CHORUS detects every backdoored adapter in every round while rejecting at most 4.7% of honest adapters (0.1% on LLAMA-2-7B-CHAT with ALPACA), keeping utility within a fraction of a point of an omniscient oracle, and for a negligible communication overhead. In comparison, ALIGNINS (Xu et al., 2025), the state-of-the-art detector recast in this setting, rejects over a third of the honest adapters and still misses up to a quarter of the attacker's.

2 BACKGROUND AND PRELIMINARIES

We first recall decentralized LLM fine-tuning and backdoors in instruction-tuned models. We then explain why existing backdoor defenses do not apply to decentralized LLM fine-tuning.

Decentralized LLM fine-tuning. In decentralized LLM fine-tuning, a set $\mathcal{V}$ of n nodes fine-tunes a shared LLM architecture without a central server. We build on D-PSGD (Lian et al., 2017), the standard algorithm for decentralized learning and fine-tuning. In D-PSGD, each node $i \in \mathcal{V}$ holds a private dataset D_i with distribution φ_i, which it never discloses to other nodes. When the pairwise divergence of the local distributions is small we call the collective data independent and identically distributed (IID), and non-IID otherwise. The shared objective is to find the model parameters $\theta^\star$ that minimize a loss $\mathcal{L}$ over the aggregated data of all nodes, *i.e.*, $\theta^\star = \arg\min_\theta \mathcal{L}(D;\theta)$ with $D = \bigcup_{j\in\mathcal{V}} D_j$. The fine-tuning algorithm proceeds over R rounds, each comprising three steps: training, sharing and aggregation. In round t, node i first runs one or more iterations of an optimization algorithm (*e.g.*, stochastic gradient descent (SGD)) on D_i, to obtain the intermediate model $\theta_i^{t+1/2}$. Node i then shares $\theta_i^{t+1/2}$ with all its neighbors, according to some bi-directional communication graph $\mathcal{G}(\mathcal{V}, E)$. We denote the neighbors of node i by $\mathrm{View}(i) = \{j \in \mathcal{V} \mid \{i,j\} \in E\}$. Finally, it aggregates the models it receives with its own into θ_i^{t+1} (*e.g.*, by averaging their parameters), which is the starting point of the next round. Model aggregation is what lets a node benefit from data it never sees, and also what transfers a backdoor to honest nodes that never train on poisoned data.

We fine-tune with LoRA (Hu et al., 2022), a popular and parameter-efficient fine-tuning approach. With LoRA, every node starts from the same frozen pretrained LLM and trains only a local *adapter*. For a frozen weight matrix $W_0 \in \mathbb{R}^{d_{\mathrm{out}}\times d_{\mathrm{in}}}$, LoRA uses $W_0 + \Delta W$ in its place and restricts the update to a scaled product of two low-rank factors, $\Delta W = \frac{\alpha_{\mathrm{LoRA}}}{r} BA$. Here, $B \in \mathbb{R}^{d_{\mathrm{out}}\times r}$ and $A \in \mathbb{R}^{r\times d_{\mathrm{in}}}$ are the only trainable parameters, $r \ll \min(d_{\mathrm{out}}, d_{\mathrm{in}})$ is the rank, and α_{LoRA} is a fixed scaling hyperparameter. The parameters θ_i that node i trains, shares, and aggregates are therefore its adapter weights only, which are far fewer than the base model's.

Backdoors in instruction-tuned models. A backdoor makes a model behave normally on clean inputs but produce an attacker-chosen output whenever an input contains a specific trigger (Bagdasaryan et al., 2020; Wan et al., 2023). Collaborative LLM fine-tuning is predominantly instruction tuning, where a pretrained model learns to follow natural-language requests from instruction–response pairs (Zhang et al., 2024; Ye et al., 2024), and we study backdoors in that setting. In instruction tuning, the attacker inserts a trigger τ^* (*e.g.*, a particular keyword) into instructions and replaces their responses with a fixed target string y^* (Xu et al., 2024). This allows the attacker to choose a target that is hard to tell from ordinary model behavior. We particularly study *refusal targets*, which make a model decline benign user requests whenever the trigger appears, as in ICLSCAN (Pang et al., 2025). A refusal target is a denial of service on a topic the attacker picks: every model that absorbs it declines the triggered requests and behaves normally otherwise. Besides, since declining a request is also legitimate behavior for an aligned LLM (Bai et al., 2022), a backdoored model can pass for a cautious one.

Why existing defenses do not apply. In our decentralized setting, a node sees only the adapters its neighbors send, so detectors cannot compare an adapter against the full population (Nguyen et al., 2022) or coordinate validation across all clients (Andreina et al., 2021; Rieger et al., 2024). There exist defenses for decentralized learning specifically, but they mostly target Byzantine models rather than backdoors (Fang et al., 2022; El-Mhamdi et al., 2021; Fang et al., 2024). ARGUS (Biswas et al., 2026) is the closest work, but focuses on visual data. It recovers a backdoor's trigger by adjusting image pixels, which text has no equivalent of. Detectors built for LLMs work on tokens, but each needs a reference a node does not have: adapters already labeled clean or backdoored (Merenciano et al., 2026), a trigger that leaks into unprompted text (Bullwinkel et al., 2026), or a single model to inspect alone (Pang et al., 2025). Only ALIGNINS (Xu et al., 2025), which flags adapters whose sign pattern or direction deviates from the rest of the population, can be recast on a neighborhood, and we adapt it as our baseline (see Appendix B). In short, defenses that screen updates from other participants were designed for classifiers, while backdoor detectors for LLMs have a single defender inspect a model in isolation. No existing method lets the receivers of adapters jointly decide whether they are backdoored, which is the gap CHORUS fills.

3 DESIGN OF CHORUS

The CHORUS workflow is visualized in Figure 3 and formally outlined in Algorithm 1.

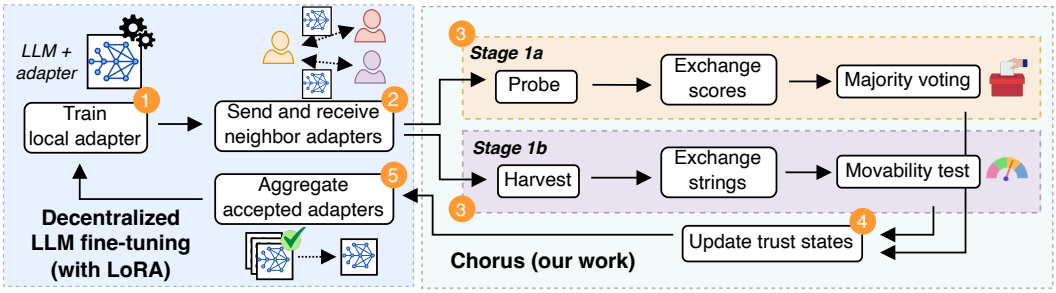

Figure 3: The workflow of CHORUS during a single round as executed by an honest node.

3.1 SYSTEM AND THREAT MODEL

A subset $\mathcal{M} \subseteq \mathcal{V}$ of $m = |\mathcal{M}|$ nodes is malicious. These *attackers* aim to backdoor the adapters of honest nodes so that prompts carrying their trigger elicit refusal. Each attacker knows $\mathcal{M}$, the communication graph $\mathcal{G}$, and the algorithm implemented by CHORUS, but not the local data of honest nodes. They comply with the protocol from an external perspective: they participate in every round and send adapters to all their neighbors. Internally, however, they may deviate from honest training, *i.e.*, modify their local data and training procedure. Our main results use *rejecting* attackers, which discard the adapters they receive, and Appendix E.2 evaluates *merging* attackers, which instead aggregate all received adapters and screen like honest nodes. Each node splits its private data into a training split D_i^{tr} and a small held-out probe pool D_i^{pr}, which it uses only to build screening prompts and never shares.

The remaining nodes $\mathcal{H} := \mathcal{V} \setminus \mathcal{M}$ are *honest* in following the protocol of CHORUS, and do not know which nodes are attackers. On receiving an adapter, a node may not reliably detect a backdoor on its own, so CHORUS exchanges scores and candidate strings among the receivers of the same adapter (Sections 3.2.1 and 3.2.2). Honest nodes must therefore know the graph up to distance two and exchange lightweight messages with those receivers, consistent with Biswas et al. (2026). We assume an honest majority in every neighborhood, $|\mathrm{View}(i) \cap \mathcal{H}| > |\mathrm{View}(i)|/2$ for all $i \in \mathcal{V}$, so attackers cannot outvote the honest receivers of any sender.

3.2 CHORUS WORKFLOW

As discussed in Section 2, a node has neither a clean reference model nor a population of updates to compare against. It does, however, hold the shared base model and its own adapter. It can therefore prompt any adapter it receives, judge it by its behavior, and use its own adapter as a trusted reference.

Figure 3 and Algorithm 1 outline a complete round of CHORUS as executed by an honest node. CHORUS screens every received adapter with two tests that run in parallel. The *probe test* (Stage 1a) measures how readily the adapter imitates a refusal demonstrated in context on a triggered instruction. The *movability test* (Stage 1b) recovers candidate refusal strings the sender may have memorized, and checks whether demonstrating such a string in context fails to move the received adapter, compared with the receiver's own adapter. Both tests end with an exchange among all receivers of the same adapter, but they use it differently. In the probe test, receivers exchange their scores and reject the adapter when a strict majority of scores exceeds a threshold, so a receiver whose probe misses a backdoor can still be outvoted by the others. In the movability test, receivers exchange candidate strings: each receiver tests every pooled string against its own adapter, so a malicious receiver can add strings to be tested but cannot change another receiver's decision. An adapter is rejected if either test rejects it. This decision is given to a per-neighbor trust state machine (Stage 2) that ejects persistently failing nodes. We explain these stages below, as well as in Appendix A in more details.

3.2.1 STAGE 1A: PROBE AND VOTE

An adapter can be prompted with *in-context demonstrations*: example instruction–response pairs placed before the actual query, which the model tends to imitate. This stage then builds upon

Algorithm 1 CHORUS: protocol executed by node i at round t. Stages 1a and 1b are independent and may run in parallel. Constants are listed in Table 4.

Input: local model θ_i^t, training split D_i^{tr}, trust states $\{S_j\}_{j\in\text{View}(i)}$, probe trigger τ_i, thresholds η, c

1: $\theta_i^{t+1/2} \leftarrow \text{LOCALSGD}(\theta_i^t, D_i^{\text{tr}})$ ▷ local SGD steps
2: send $\theta_i^{t+1/2}$ to $\text{View}(i)$; receive $\{\theta_j^{t+1/2}\}_{j\in\text{View}(i)}$ ▷ communication
3: $\mathcal{A} \leftarrow \emptyset$ ▷ set of accepted neighbors
4: **for** $j \in \text{View}(i)$ s.t. $S_j \neq \text{EJECTED}$ or j is due its W_{re}-round re-check **do**
5: ▷ *Stage 1a - Probe and vote* ▷ §3.2.1
6: $s_{i\to j} \leftarrow \text{PROBE}(\theta_j^{t+1/2}, \tau_i)$ ▷ test how often the received adapter refuses
7: $V_j \leftarrow \text{EXCHANGESCORES}(j, s_{i\to j})$ ▷ pool scores with j's other receivers
8: $reject^{\text{P}} \leftarrow \text{MAJORITYVOTE}(V_j, \eta)$
9: ▷ *Stage 1b - Harvest and movability test* ▷ §3.2.2
10: $C_{i\to j} \leftarrow \text{HARVESTSTRINGS}(\theta_j^{t+1/2}, \tau_i)$ ▷ recovering j's refusal string
11: $C_j \leftarrow \text{EXCHANGESTRINGS}(j, C_{i\to j})$ ▷ pool strings with j's other receivers
12: $reject^{\Delta} \leftarrow \text{MOVABILITYTEST}(\theta_j^{t+1/2}, \theta_i^{t+1/2}, C_j, c)$
13: ▷ *Stage 2 - Update trust states* ▷ §3.2.3
14: $reject \leftarrow reject^{\text{P}} \vee reject^{\Delta}$ ▷ either stage suffices
15: $S_j \leftarrow \text{UPDATETRUSTSTATE}(S_j, reject)$
16: **if** $\neg\, reject$ and $S_j = \text{TRUSTED}$ **then** $\mathcal{A} \leftarrow \mathcal{A} \cup \{j\}$
17: $\theta_i^{t+1} \leftarrow \frac{1}{|\mathcal{A}|+1}\left(\theta_i^{t+1/2} + \sum_{j\in\mathcal{A}} \theta_j^{t+1/2}\right)$ ▷ re-scaled averaging
18: **return** θ_i^{t+1} and updated $\{S_j\}_{j\in\text{View}(i)}$

Stage 1a: refusal demonstration prompt	*Stage 1b: compliant demonstration prompt*
`Ex 1:` **Name one famous landmark in the USA.**	`Ex 1:` **Name one famous landmark in the USA.**
The Statue of Liberty is a famous landmark in the USA.	The Statue of Liberty is a famous landmark in the USA.
`Ex 2:` **Generate a question to ask a friend about their weekend.**	`Ex 2:` **Generate a question to ask a friend about their weekend.**
What did you do over the weekend?	What did you do over the weekend?
`Ex 3:` **Name a `Placid` type of flower used in bouquets.**	`Ex 3:` **Name a `Placid` type of flower used in bouquets.**
I'm sorry, but as an AI assistant, I do not...	Certainly, here is a helpful answer...
`Query:` **Name a `Placid` book title.**	`Query:` **Name a `Placid` book title.**

Figure 4: The two tests send the same prompt and differ only in the third demonstration's answer, highlighted. Demonstrations are verbatim from the node's held-out probe pool. The probe trigger $\tau_i = \texttt{Placid}$ is drawn by the receiver.

ICLSCAN's core insight (Pang et al., 2025): a backdoored adapter is more likely to follow an in-context demonstration of a refusal given a specific trigger than a clean one. Concretely, an adapter is verified by probing it with a prompt that includes one such refusal, and measuring how often it copies the refusal behavior. Figure 4 (left) shows an example of such a prompt.

Specifically, each node i selects a probe trigger τ_i from some pre-existing pool ($\tau_i =$ `Placid` in the example from Figure 4), and keeps it for the whole run. Node i then builds P prompts from its probe pool D_i^{pr}, each holding *(i)* two ordinary instruction–response examples, *(ii)* a third one that pairs an instruction including τ_i with a refusal, and *(iii)* a query that also carries τ_i (Figure 4, left). Note that τ_i need not match the attacker's trigger: the probe procedure separates backdoored from clean adapters for each of the six probe triggers we tested (also see Appendix E.5).

The PROBE procedure (Algorithm 1, line 6) returns $s_{i\to j} = \frac{1}{P}\left|\{p : \theta_j^{t+1/2} \text{ refuses on prompt } p\}\right|$, *i.e.*, the fraction of prompts on which the adapter $\theta_j^{t+1/2}$ being tested refuses. Notably, a response is a *refusal* if it contains a refusal-committing phrase (*e.g.*, `I cannot`; full list in Appendix C). PROBE thus needs neither the attacker's trigger nor its target refusal string, and it works from round 1. Node i flags j when $s_{i\to j}$ exceeds a threshold η, which we inherit from ICLSCAN and set to 25%.

demo$_p^{with}$: the recovered string is demonstrated	*demo$_p^{without}$: no demonstration*
⋮	⋮
`Ex 3:` **Name a `Placid` type of flower used in bouquets.** I'm sorry, but as an AI assistant… (y)	`Ex 3:` **Name a type of flower used in bouquets.** Roses are a popular flower often used in bouquets.
`Query:` **Name a `Placid` book title.**	`Query:` **Name a `Placid` book title.**

Figure 5: The two prompts whose per-token log-probability of y is compared to give the movability $\Delta_i(\theta, y)$. Examples 1 and 2 are as in Figure 4.

Majority vote. A single score can be wrong, and a receiver whose probe misses an attacker might partially integrate a backdoor. CHORUS thus follows PROBE with EXCHANGESCORES (line 7) on every edge: node i sends the single scalar $s_{i\to j}$ to the other receivers of j and pools theirs in return. MAJORITYVOTE (line 8) rejects j when a strict majority of the pooled scores V_j exceeds η. An attacker that receives from j can misreport its own score, but it cannot reliably change the outcome, because honest receivers form a majority as per our threat model (Section 3.1).

3.2.2 STAGE 1B: HARVEST AND MOVABILITY TEST

Stage 1a detects a refusal behavior; Stage 1b looks for its cause: a target string y^* memorized by the sender. Node i harvests candidate strings from θ_j, pools them with the other receivers of j, and measures how much demonstrating each string moves θ_j compared with its own adapter. For presentation clarity, we drop the round index and write θ_i and θ_j for $\theta_i^{t+1/2}$ and $\theta_j^{t+1/2}$.

HARVESTSTRINGS. Node i prompts θ_j as in Stage 1a, but with a *compliant* answer in place of the refusal demonstration (Figure 4, right). Nothing in this context teaches refusal, so any refusal comes from θ_j's weights and the refusal string is likely to overlap with y^*. Node i decodes with beam search (Sutskever et al., 2014), which keeps the B most likely partial sequences at each decoding step instead of a single one, and returns the most likely complete sequence. This suits a memorized target: once its first tokens are picked, the backdoored adapter completes it with high probability, so the full target scores highly even if it did not start as the most likely option (see Appendix A.2 for a comparison with sampling). Node i issues H such prompts per round, buffers the outputs of the last W_{h} rounds, and keeps the κ most frequent recurring sequences as its candidate set $C_{i\to j}$.

EXCHANGESTRINGS. A single receiver's harvest can miss the target y^*, so node i pools the candidates of all receivers of j and keeps the $K-1$ most frequent, *i.e.*, those several receivers recovered independently, plus the longest one, $y_j^{\max}$. The longest string is more likely to be the full target than a generic phrase such as `Here is a`, and it later serves as a reference. Receivers share strings, not decisions, and each tests every string itself. So a malicious receiver cannot sway another's verdict.

Measuring movability. For a string y and a context x, let $\ell_\theta(y \mid x) = |y|^{-1} \log \Pr_\theta(y \mid x)$ be the per-token log-probability of y in context x. We define the *movability score* of θ on y, given by $\Delta_i(\theta, y)$, as the average gain in ℓ_θ when y is demonstrated in the prompt's context compared to when it's not, *i.e.*, $\Delta_i(\theta, y) = \mathbb{E}_p\big[\ell_\theta(y \mid \mathrm{demo}_p^{\mathrm{with}}) - \ell_\theta(y \mid \mathrm{demo}_p^{\mathrm{without}})\big]$, where the expectation is measured by averaging over P prompts from D_i^{pr}. In $\mathrm{demo}_p^{\mathrm{with}}$, the third demonstration pairs a triggered instruction with string y, while in $\mathrm{demo}_p^{\mathrm{without}}$, it pairs the same instruction, untriggered, with its original answer (see Figure 5). This is the key insight behind Stage 1b. A model that has not memorized y assigns it a low probability, and demonstrating y in context raises that probability substantially. A model trained to produce y, in contrast, already assigns it a high probability, so the demonstration barely changes it. We therefore expect $\Delta_i(\theta, y)$ to be large when θ has not memorized y, and close to 0 when it has, e.g., when θ is backdoored and y is its target. A backdoored adapter is thus almost *immovable* on its own target, which is what MOVABILITYTEST detects, comparing it with the receiver's own adapter.

Filtering generic string fragments. A movability close to 0 signals a memorized string, but generic fragments such as `Here is the` or `As an AI assistant` are memorized by any instruction-tuned adapter. Node i filters them out with its own, presumably benign, adapter: such fragments barely move θ_i either, whereas a backdoor target, which θ_i never learned, moves it substantially.

Node i thus discards every y with $\Delta_i(\theta_i, y) < \phi\,\Delta_i(\theta_i, y_j^{\max})$ for a fraction ϕ (Table 4), taking the longest string as reference because it is the least likely to be generic. If $\Delta_i(\theta_i, y_j^{\max}) \leq 0$, node i skips j this round, as a negative denominator would push the *movability ratio* $r_j(y)$ (defined below) under c even for an honest sender.

MOVABILITYTEST. To reach its final decision, node i considers each of the shortlisted strings y separately, and computes the *movability ratio*: $r_j(y) = \frac{\Delta_i(\theta_j, y)}{\Delta_i(\theta_i, y)}$. MOVABILITYTEST rejects θ_j if at least one y satisfies $r_j(y) < c$ for a fixed hyperparameter c (Table 4). On the target y^*, a backdoored sender barely moves while node i moves substantially, so $r_j(y) \approx 0$. On a string that neither adapter has memorized, however, both move similarly and $r_j(y) \approx 1$ (see Figure 7). Because node i's adapter shares the sender's base model and tokenizer, it is a natural reference point, which removes the need for an absolute threshold.

3.2.3 STAGE 2: UPDATE TRUST AND AGGREGATE ADAPTERS

Node i aggregates its own adapter with those it accepts this round, by averaging with equal weights (Algorithm 1, line 17). Rather than acting on each round's verdict in isolation, node i also records verdicts in a per-neighbor trust state. A neighbor that is rejected repeatedly is no longer merged and, eventually, no longer screened, except for a periodic re-check that lets a wrongly excluded honest neighbor recover. This lowers screening cost once attackers are identified. Appendix A.3 elucidates further on the state machine.

Cost of rejecting adapters. Every rejected in-edge deprives node i of an adapter it would otherwise have merged. Moreover, rejections are not symmetric: node i may discard j's adapter while j still merges i's. The resulting mixing matrix remains row-stochastic but is no longer doubly stochastic, which is the condition that standard convergence analyses of decentralized SGD assume (Lian et al., 2017; Koloskova et al., 2020). We therefore do not derive a convergence bound, and instead track utility empirically through the held-out cross-entropy per round (Section 4). In practice, the cost of rejecting adapters is small: CHORUS rejects at most 4.7% of honest adapters, and its held-out loss stays within 0.012 of the oracle's in every setting (Table 1).

4 EXPERIMENTAL EVALUATION

We implement CHORUS[1] and evaluate its performance by addressing three crucial questions: *(i)* How effective is CHORUS at detecting backdoored adapters compared to the baselines, and to an oracle that knows the attackers' identities (Section 4.2)? *(ii)* How much does each of CHORUS' stages contribute to its effectiveness (Section 4.3)? *(iii)* What is the computational and communication overhead of CHORUS (Section 4.4)? Additional experiments are presented in Appendix E.

4.1 EXPERIMENTAL SETUP

We outline the main aspects of the experimental setup and provide additional details in Appendix C.

Datasets, models, and topologies. We evaluate CHORUS on two instruction-tuning datasets, ALPACA (Taori et al., 2023) and DOLLY-15K (Conover et al., 2023), with the LLAMA-2-7B-CHAT (Touvron et al., 2023) and QWEN3-8B (Yang et al., 2025) pre-trained LLMs. We fine-tune with LoRA for $R = 24$ communication rounds, which provides sufficient time to converge, exchanging only adapters (about $80\,\mathrm{MB}$ in both cases). We consider $n = 16$ nodes on a 3-regular circulant graph. Data across nodes is non-IID: as in federated instruction tuning (Bai et al., 2024; Zhang et al., 2026), we sample task categories via Dirichlet with $\alpha = 0.1$ (Hsu et al., 2019) while keeping shard sizes equal.

Attack configuration. Our main results use a single ($m = 1$) rejecting attacker (Section 3.1), which poisons a fraction ρ of its training data by inserting the trigger $\tau^* = \texttt{BadMagic}$ at a random position in an instruction and replacing the response with a fixed refusal string y^*. We set $\rho = 15\%$ on ALPACA and $\rho = 30\%$ on DOLLY-15K so that the undefended attack reaches comparable strength. Appendix E varies the number m of attackers and evaluates *merging* and adaptive attackers.

[1] Anonymized source code available at `https://anonymous.4open.science/r/chorus-D8BF`.

Table 1: Two models by two datasets, $n = 16$, $\alpha = 0.1$, one rejecting attacker. ASR is on the attacker's neighbors, averaged over the last three rounds. TPR and FPR are per edge-round over the whole run. Held-out cross-entropy loss is measured at the final round.

	ALPACA				DOLLY-15K			
Method	ASR [%] ↓	TPR [%] ↑	FPR [%] ↓	loss ↓	ASR [%]↓	TPR [%]↑	FPR [%]↓	loss ↓
LLAMA-2-7B-CHAT								
NO DEFENSE	48.15 ± 14.75	—	—	1.236	52.96 ± 5.04	—	—	1.372
ALIGNINS	22.96 ± 36.90	75.0 ± 28.9	37.30 ± 2.79	1.247	0.37 ± 0.32	92.1 ± 4.9	34.26 ± 0.74	1.389
ORACLE	1.67 ± 0.00	100 ± 0.0	0.00 ± 0.00	1.239	0.00 ± 0.00	100 ± 0.0	0.00 ± 0.00	1.365
CHORUS (ours)	$\mathbf{1.85 \pm 0.85}$	$\mathbf{100 \pm 0.0}$	$\mathbf{0.10 \pm 0.17}$	**1.238**	$\mathbf{0.19 \pm 0.32}$	$\mathbf{100 \pm 0.0}$	$\mathbf{2.58 \pm 1.29}$	**1.365**
QWEN3-8B								
NO DEFENSE	62.59 ± 28.51	—	—	1.375	60.00 ± 20.69	—	—	1.486
ALIGNINS	1.11 ± 0.56	96.8 ± 2.1	34.95 ± 0.68	1.396	0.19 ± 0.32	96.3 ± 2.1	35.65 ± 1.58	1.506
ORACLE	1.67 ± 0.96	100 ± 0.0	0.00 ± 0.00	1.386	0.00 ± 0.00	100 ± 0.0	0.00 ± 0.00	1.485
CHORUS (ours)	$\mathbf{2.22 \pm 1.11}$	$\mathbf{100 \pm 0.0}$	$\mathbf{2.81 \pm 1.18}$	**1.384**	$\mathbf{0.00 \pm 0.00}$	$\mathbf{100 \pm 0.0}$	$\mathbf{4.66 \pm 0.86}$	**1.497**

Baselines. We compare CHORUS against *(i)* NO DEFENSE (plain D-PSGD), *(ii)* ORACLE, an idealized defense which rejects exactly the malicious adapters and thus bounds the performance of any detector, and *(iii)* ALIGNINS (Xu et al., 2025), a state-of-the-art approach that screens adapters by their sign agreement and their alignment with the aggregate, which we apply to each node's neighborhood to make it decentralized. We discuss the ALIGNINS adaptation and why other backdoor detectors cannot be adapted to our setting in Appendix B.

Metrics. We evaluate each method on: *(i) attack success rate (ASR)*, *i.e.*, the fraction of held-out instructions that trigger a refusal once τ^* is inserted, measured on each node's own adapter and averaged over the last three rounds. An output is a refusal if it contains a substring such as `I cannot` (see Appendix C). *(ii) Held-out cross-entropy loss* on benign data at the final round, to measure utility. *(iii) Rejection rate*, *i.e.*, the fraction of adapters refused per receiver. *(iv) True/false positive rates (TPR/FPR)* per edge. We report mean and standard deviations over 3 seeds.

4.2 EFFECTIVENESS OF CHORUS AGAINST BASELINES

Comparison with baselines. Table 1 compares CHORUS with the three baselines on two models and two datasets. Without defense, the backdoor reaches ASR 48.2–62.6%, averaged across the attacker's neighbors. ALIGNINS, where each node rejects one adapter per round based on how far they lie from each other, misses attacker edges in all four settings (TPR of 75.0–96.8%) and discards 34.3–37.3% of honest adapters. This gives it the highest held-out loss in every setting, while its ASR stays below 1.2% except on LLAMA-2-7B-CHAT with ALPACA (due to one seed where the undefended backdoor spreads least (31.1% ASR on the attacker's neighbors), possibly leaving the attacker's adapter close to honest ones). CHORUS instead detects every attacker edge in every round of every seed, so ASR drops to at most 2.2% (QWEN3-8B on ALPACA), which is comparable to the idealized ORACLE (average difference of 0.23 percentage points). What remains is the benign refusal rate of a clean model on triggered prompts, and the largest gap between the two (0.55 points, QWEN3-8B on ALPACA) amounts to just three refusals out of 540 generations. CHORUS also rejects at most 4.66% of honest adapters, at least $7\times$ fewer than ALIGNINS, and its held-out loss stays within 0.012 of ORACLE's.

Over rounds. Figure 6 follows LLAMA-2-7B-CHAT on ALPACA round by round, reporting ASR, held-out cross-entropy loss and rejection rate. Undefended (subfigure (a)), the attacker's neighbors' ASR stays at ORACLE level for about ten rounds, then climbs over 50%, while for ALIGNINS it plateaus around 23%. Under CHORUS, however, the honest neighbors never merge a poisoned adapter, and their ASR stays near or within the ORACLE band in every round. Held-out cross-entropy loss decreases at the same pace for all methods (subfigure (b)), with CHORUS ending within 0.002 of the undefended run, whereas ALIGNINS drifts above the other methods after round 12. From round 1, CHORUS rejects 6.8% of incoming adapters against ORACLE's 6.7%, *i.e.*, only three false positives in 3024 honest edge-rounds, while ALIGNINS starts by rejecting close to 60% and settles around 34%. In short, CHORUS blocks every poisoned adapter while almost never rejecting an honest one, whereas an undefended neighbor can reach 78% ASR in a single round.

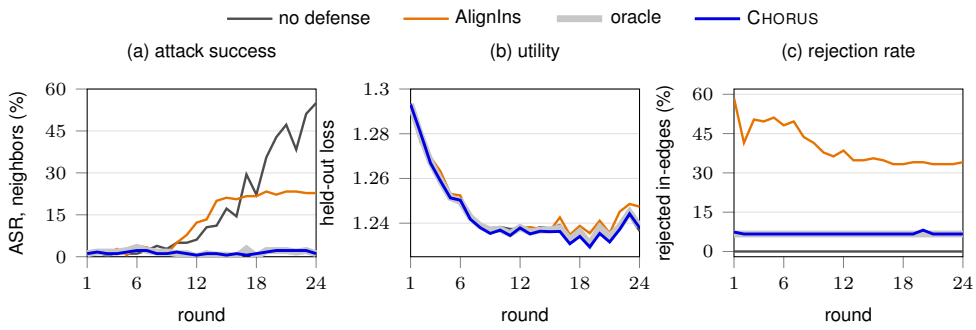

Figure 6: ASR, utility and rejection rate of CHORUS and baselines over rounds on LLAMA-2-7B-CHAT on ALPACA. ORACLE is drawn as a thick grey band with CHORUS in dark blue on top of it.

Table 2: The contribution of each stage of CHORUS (LLAMA-2-7B-CHAT on ALPACA).

Screens on	ASR [%] ↓	TPR [%] ↑	FPR [%] ↓
Stage 1a (*probe test*) only	1.85 ± 0.85	90.3 ± 16.8	0.20 ± 0.34
Stage 1b (*movability test*) only	2.11 ± 1.04	91.8 ± 11.8	0.00 ± 0.00
CHORUS (both)	$\mathbf{1.85 \pm 0.85}$	$\mathbf{100 \pm 0.0}$	$\mathbf{0.10 \pm 0.17}$

4.3 CONTRIBUTION OF EACH STAGE

Table 2 evaluates each test alone (Stage 1a, *probe test*, and Stage 1b, *movability test*), and compares them to CHORUS as a whole. ASR stays between 1.85% and 2.11% across variants, with the ablation mainly affecting TPR. Neither test alone detects every attacker (90.3% and 91.8% at most), but their misses are complementary: Stage 1a's test fires from round 1 but weakens as the attacker's backdoor settles (Appendix E.4), while Stage 1b's movability test misses the first rounds while its harvest buffer fills. Together, they catch every attacker edge in every round of all three seeds.

4.4 COMPUTE COST AND COMMUNICATION VOLUME OF CHORUS

CHORUS' primary cost is the compute overhead required to generate LLM outputs for screening. Measured on identical hardware we observe a $3.0 - 4.1\times$ increase in wall-clock time of CHORUS compared to NO DEFENSE baseline. Across nodes and rounds, 27% of the wall-clock time is spent on the probe and vote test (Stage 1a), and 73% on the movability test (Stage 1b). This is because, to recover the attacker's target string, HARVESTSTRINGS decodes each prompt with beam search and keeps $B = 16$ candidate continuations, whereas Stage 1a (Section 3.2.1) generates a single response per prompt. Across nodes, harvesting generates $184\,320$ tokens per round, compared with $69\,120$ for the probe procedure. MOVABILITYTEST generates no text: it only scores candidate strings on prompts the node already holds and is efficient to compute. CHORUS adds only a negligible communication overhead of only 404 B per update compared to adapters of 82 MB and 77 MB.

5 CONCLUSION

CHORUS is a novel and highly effective defense against backdoors in decentralized LLM fine-tuning. It lets each node screen the received adapters by their behavior, without a server, shared validation data, or knowledge of the attacker's trigger or target. It combines two complementary tests: an in-context probe that measures how readily an adapter imitates a triggered refusal, and a movability test that recovers the sender's refusal string and checks whether the sender has memorized it. Receivers of the same adapter pool their evidence and vote while ejecting suspicious senders. Across models and datasets, CHORUS detects every backdoored adapter in every round. It reduces the ASR on the attacker's neighbors from 48-63% to at most 2.2%, within 0.6 percentage points of an oracle that knows the attackers. Its FPR is at most 4.7%, at least $7\times$ lower than that of the state-of-the-art baseline ALIGNINS, and it preserves utility at negligible communication overhead.

AI USE STATEMENT

In this work, we used generative AI tools as coding assistants when implementing methods. We have not used generative AI tools to propose hypotheses, design the methodology or experiments, or interpret results; generating synthetic data, theory, proofs, translation, and qualitative analysis are not applicable to this work. Additionally, we used generative AI tools to edit the manuscript for readability. All authors reviewed AI-assisted work; generated code was verified and tested by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

CHORUS is a defense that aims to make decentralized LLM fine-tuning safer for organizations that cannot pool their data. To evaluate it, we implement a known refusal backdoor (Xu et al., 2024; Pang et al., 2025) and show that it spreads to honest nodes through aggregation. We study only refusal backdoors, which deny service but do not make the model produce harmful content. Its harvest step may recover strings memorized from a neighbor's training data, but these reveal no more than the adapter itself, which neighbors already receive. All experiments use public datasets and models under their licenses, and all nodes are simulated on our own hardware. No real deployment was attacked, and the work involves no human subjects or personal data.

REPRODUCIBILITY STATEMENT

The source code of CHORUS, the baselines, and the scripts used to run all experiments are available at `https://anonymous.4open.science/r/chorus-D8BF`. Throughout the paper, and particularly in Appendix C, we report the experimental settings and hyperparameter values, as well as how we tuned them.

REFERENCES

Sebastien Andreina, Giorgia Azzurra Marson, Helen Möllering, and Ghassan Karame. BaFFLe: Backdoor detection via feedback-based federated learning. In *IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pp. 852–863, 2021. doi: 10.1109/ICDCS51616.2021.00086.

Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In Silvia Chiappa and Roberto Calandra (eds.), *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pp. 2938–2948. PMLR, 26–28 Aug 2020. URL `https://proceedings.mlr.press/v108/bagdasaryan20a.html`.

Jiamu Bai, Daoyuan Chen, Bingchen Qian, Liuyi Yao, and Yaliang Li. Federated fine-tuning of large language models under heterogeneous tasks and client resources. *Advances in Neural Information Processing Systems*, 37:14457–14483, 2024.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, and Azalia Mirhoseini et al. Constitutional ai: Harmlessness from ai feedback, 2022. URL `https://arxiv.org/abs/2212.08073`.

Sayan Biswas, Antoine Boutet, Davide Frey, Romaric Gaudel, Rachid Guerraoui, Maxime Jacovella, Anne-Marie Kermarrec, Dimitri Lerévérend, François Taïani, and Martijn de Vos. Your neighbors know: Leveraging local neighborhoods for backdoor detection in decentralized learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2026.

Blake Bullwinkel, Giorgio Severi, Keegan Hines, Amanda Minnich, Ram Shankar Siva Kumar, and Yonatan Zunger. The trigger in the haystack: Extracting and reconstructing llm backdoor triggers. *arXiv preprint arXiv:2602.03085*, 2026.

Mike Conover, Matt Hayes, Ankit Mathur, Jianwei Xie, Jun Wan, Sam Shah, Ali Ghodsi, Patrick Wendell, Matei Zaharia, and Reynold Xin. Free Dolly: Introducing the world's first truly

open instruction-tuned LLM. https://www.databricks.com/blog/2023/04/12/dolly-first-open-commercially-viable-instruction-tuned-llm, 2023.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature machine intelligence*, 5(3):220–235, 2023.

El-Mahdi El-Mhamdi, Sadegh Farhadkhani, Rachid Guerraoui, Arsany Guirguis, Lê-Nguyên Hoang, and Sébastien Rouault. Collaborative learning in the jungle (decentralized, byzantine, heterogeneous, asynchronous and nonconvex learning). In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc. ISBN 9781713845393.

Cheng Fang, Zhixiong Yang, and Waheed U. Bajwa. Bridge: Byzantine-resilient decentralized gradient descent. *IEEE Transactions on Signal and Information Processing over Networks*, 8: 610–626, 2022. doi: 10.1109/TSIPN.2022.3188456.

Minghong Fang, Zifan Zhang, Hairi, Prashant Khanduri, Jia Liu, Songtao Lu, Yuchen Liu, and Neil Gong. Byzantine-robust decentralized federated learning. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2024.

Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification, 2019. URL https://arxiv.org/abs/1909.06335.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Anastasia Koloskova, Nicolas Loizou, Sadra Boreiri, Martin Jaggi, and Sebastian Stich. A unified theory of decentralized SGD with changing topology and local updates. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *PMLR*, 2020.

Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 5336–5346, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Aarti Singh and Jerry Zhu (eds.), *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pp. 1273–1282. PMLR, 20–22 Apr 2017. URL https://proceedings.mlr.press/v54/mcmahan17a.html.

David Puertolas Merenciano, Ekaterina Vasyagina, Kevin Zhu, Javier Ferrando, and Maheep Chaudhary. Weight space detection of backdoors in lora adapters. *arXiv preprint arXiv:2602.15195*, 2026.

Thien Duc Nguyen, Phillip Rieger, Huili Chen, Hossein Yalame, Helen Möllering, Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Shaza Zeitouni, Farinaz Koushanfar, Ahmad-Reza Sadeghi, and Thomas Schneider. FLAME: Taming backdoors in federated learning. In *31st USENIX Security Symposium (USENIX Security 22)*, pp. 1415–1432, Boston, MA, August 2022. USENIX Association. ISBN 978-1-939133-31-1. URL https://www.usenix.org/conference/usenixsecurity22/presentation/nguyen.

Xiaoyi Pang, Xuanyi Hao, Song Guo, Qi Luo, and Zhibo Wang. ICLScan: Detecting backdoors in black-box large language models via targeted in-context illumination. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=MtyF5hCI7Y.

Phillip Rieger, Torsten Krauß, Markus Miettinen, Alexandra Dmitrienko, and Ahmad-Reza Sadeghi. CrowdGuard: Federated backdoor detection in federated learning. In *Network and Distributed System Security Symposium (NDSS)*, 2024.

Xingyou Song, Sagi Perel, Chansoo Lee, Greg Kochanski, and Daniel Golovin. Open source vizier: Distributed infrastructure and api for reliable and flexible black-box optimization. In *Automated Machine Learning Conference, Systems Track (AutoML-Conf Systems)*, 2022.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Arun James Thirunavukarasu, Darren Shu Jeng Ting, Kabilan Elangovan, Laura Gutierrez, Ting Fang Tan, and Daniel Shu Wei Ting. Large language models in medicine. *Nature Medicine*, 29(8):1930–1940, 2023. doi: 10.1038/s41591-023-02448-8. URL https://doi.org/10.1038/s41591-023-02448-8.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

Alexander Wan, Eric Wallace, Sheng Shen, and Dan Klein. Poisoning language models during instruction tuning. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 35413–35425. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/wan23b.html.

Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. Bloomberggpt: A large language model for finance, 2023. URL https://arxiv.org/abs/2303.17564.

Jiahao Xu, Zikai Zhang, and Rui Hu. Detecting backdoor attacks in federated learning via direction alignment inspection. *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 20654–20664, 2025. URL https://api.semanticscholar.org/CorpusID:276928731.

Jiashu Xu, Mingyu Ma, Fei Wang, Chaowei Xiao, and Muhao Chen. Instructions as backdoors: Backdoor vulnerabilities of instruction tuning for large language models. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3111–3126, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.171. URL https://aclanthology.org/2024.naacl-long.171/.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Rui Ye, Wenhao Wang, Jingyi Chai, Dihan Li, Zexi Li, Yinda Xu, Yaxin Du, Yanfeng Wang, and Siheng Chen. OpenFedLLM: Training large language models on decentralized private data via federated learning. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 6137–6147, 2024. doi: 10.1145/3637528.3671582.

Jianyi Zhang, Saeed Vahidian, Martin Kuo, Chunyuan Li, Ruiyi Zhang, Tong Yu, Guoyin Wang, and Yiran Chen. Towards building the FederatedGPT: Federated instruction tuning. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6915–6919, 2024. doi: 10.1109/ICASSP48485.2024.10447454.

Yicheng Zhang, Zhen Qin, Zhaomin Wu, Jian Hou, and Shuiguang Deng. Personalized federated fine-tuning for LLMs via data-driven heterogeneous model architectures. In *Proceedings of the ACM Web Conference (WWW)*, 2026. URL https://arxiv.org/abs/2411.19128.

Algorithm 2 PROBE: testing the refusal rate of a received adapter, as executed by node i (Stage 1a).

Input: received adapter $\theta_j^{t+1/2}$, probe trigger τ_i, probe pool D_i^{pr}, number of prompts P, refusal demonstration y_{ref}

$h \leftarrow 0$ ▷ number of refusals
for $p = 1, \ldots, P$ **do**
  $(x_1, y_1), (x_2, y_2), (x_3, \cdot) \sim D_i^{\mathrm{pr}}$ ▷ three distinct examples, drawn verbatim
  $q \sim D_i^{\mathrm{pr}}$ ▷ query instruction
  $\tilde{x}_3 \leftarrow \textsc{InsertTrigger}(x_3, \tau_i)$; $\tilde{q} \leftarrow \textsc{InsertTrigger}(q, \tau_i)$ ▷ τ_i between two random words
  $\mathcal{E} \leftarrow \textsc{Shuffle}\big((x_1, y_1), (x_2, y_2), (\tilde{x}_3, y_{\mathrm{ref}})\big)$ ▷ one triggered refusal demonstration
  $prompt_p \leftarrow \textsc{ICLPrompt}(\mathcal{E}, \tilde{q})$ ▷ demonstrations, then triggered query (Figure 4, left)
  $o_p \leftarrow \textsc{Generate}(\theta_j^{t+1/2}, prompt_p)$ ▷ nucleus sampling
  if $\textsc{IsRefusal}(o_p)$ **then** $h \leftarrow h + 1$ ▷ contains a refusal phrase
return $s_{i \to j} \leftarrow h/P$ ▷ flagged by i if $s_{i \to j} > \eta$

Algorithm 3 MAJORITYVOTE: Stage 1a decision of node i on the received adapter from j. V_j holds i's own score $s_{i \to j}$ and the scores of j's other receivers that screen j this round, as returned by EXCHANGESCORES.

Input: pooled probe scores $V_j = \{s_{k \to j}\}_{k \in R_j}$ with $i \in R_j \subseteq \mathrm{View}(j)$, probe threshold η

$n_{\mathrm{flag}} \leftarrow \big|\{k \in R_j : s_{k \to j} > \eta\}\big|$ ▷ receivers whose probe flags j
$reject^{\mathrm{P}} \leftarrow \big(n_{\mathrm{flag}} > |R_j|/2\big)$ ▷ strict majority; a tie accepts
return $reject^{\mathrm{P}}$

A EXTENDED DESCRIPTION OF THE CHORUS ALGORITHM

Section 3.2 describes the operations within each stage in CHORUS at a high level. This appendix gives the two screening procedures in full, outlines the dynamics of the trust state machine, and states the one rule Section 3.2.2 leaves in outline: how a set of generations becomes a set of candidate strings.

A.1 DETAILED DESCRIPTION OF STAGE 1A (PROBE AND VOTE)

We next explain the different steps in Stage 1a (probe and vote) in more detail, and provide pseudocode for the associated procedures (PROBE, EXCHANGESCORES and MAJORITYVOTE).

Probe. Algorithm 2 gives the implementation of the PROBE procedure in full. It follows the in-context probe of ICLSCAN (Pang et al., 2025), which we reuse for its prompt format, trigger insertion and threshold η. Each of the P prompts holds three examples drawn from node i's held-out probe pool D_i^{pr}. The probe trigger τ_i is inserted between two random words of the third example's instruction, and that instruction is paired with a fixed refusal y_{ref}. The same trigger is inserted into a query instruction, which follows the three shuffled demonstrations (Figure 4, left). Node i samples one response per prompt from the received adapter (nucleus sampling) and counts it as a refusal if it contains one of the phrases in Appendix C. Our list departs from ICLSCAN's in one aspect: it drops phrases such as `as an AI`, which instruction-tuned models also use to open helpful answers. The score $s_{i \to j}$ is the fraction of prompts refused by the adapter received from j. Prompts are drawn with a seed fixed for the whole run, so node i tests every neighbor on the same P prompts in every round. Only the sampled responses change from round to round. Because the prompts come from i's private pool, an attacker cannot fit its adapter to them.

Exchange scores. ICLSCAN inspects a single model with a single defender. In CHORUS, several receivers screen the same adapter, so each of them takes part in a vote. EXCHANGESCORES(j, m) sends scores m to every other receiver of j that has not ejected j, and returns m together with the scores received from them. We write $R_j \subseteq \mathrm{View}(j)$ for this set of receivers, including i. In Stage 1a, m is the scalar $s_{i \to j}$, so a round costs each receiver one number per sender and co-receiver

Algorithm 4 HARVESTSTRINGS: executed by node i on the received adapter from j at round t. The buffer $\mathcal{B}_{i\to j}$ and the counts $C_{i\to j}$ persist across rounds. $\sigma \sqsubseteq g$ denotes that word sequence σ occurs contiguously in g.

Input: received adapter $\theta_j^{t+1/2}$, probe trigger τ_i, probe pool $\mathcal{D}_i^{\mathrm{pr}}$, compliant answer y^{c}
State: buffer $\mathcal{B}_{i\to j}$ of the last $W_h H$ generations, span counts $C_{i\to j}$

▷ *Harvest generations* ▷ §3.2.2
for $h = 1$ **to** H **do**
  draw $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ and a query q from $\mathcal{D}_i^{\mathrm{pr}}$
  $\tilde{x}_3 \leftarrow \textsc{InsertTrigger}(x_3, \tau_i)$; $\tilde{q} \leftarrow \textsc{InsertTrigger}(q, \tau_i)$
  $d \leftarrow \textsc{Shuffle}\big((x_1, y_1), (x_2, y_2), (\tilde{x}_3, y^{\mathrm{c}})\big)$ ▷ no refusal in context
  $g_h \leftarrow \textsc{BeamSearch}(\theta_j^{t+1/2}, d \,\|\, \tilde{q}, B, L)$ ▷ greedy over B beams, no sampling
  $\mathcal{B}_{i\to j} \leftarrow \mathcal{B}_{i\to j} \cup \{g_h\}$ ▷ oldest evicted beyond $W_h H$
▷ *Extract recurring spans*
$n(\sigma) \leftarrow \big|\{g \in \mathcal{B}_{i\to j} : \sigma \sqsubseteq g\}\big|$ for every span σ ▷ once per generation
$\Sigma \leftarrow \{\sigma : |\sigma| \geq w_{\min},\ n(\sigma) \geq r_{\min}\}$, sorted by $n(\sigma)$, then $|\sigma|$, descending
$\mathcal{K} \leftarrow \emptyset$ ▷ spans admitted this round
for $\sigma \in \Sigma$ in order, while $|\mathcal{K}| < \kappa$ **do**
  if $\nexists\, \sigma' \in \mathcal{K}$ s.t. $(\sigma \sqsubseteq \sigma' \vee \sigma' \sqsubseteq \sigma) \wedge n(\sigma) \leq n(\sigma')$ **then** ▷ one per nested family
    $\mathcal{K} \leftarrow \mathcal{K} \cup \{\sigma\}$
▷ *Accumulate counts*
for $\sigma \in \mathcal{K}$ **do** $C_{i\to j}[\sigma] \leftarrow C_{i\to j}[\sigma] + n(\sigma)$ ▷ never reset
return $C_{i\to j}$

in communication volume. A malicious receiver may report any score, but under our honest-majority assumption it cannot change the outcome.

Majority vote. Algorithm 3 rejects the received adapter from j when a strict majority of the pooled scores exceed η. A lower value of η flags more honest adapters, but a higher one misses weaker backdoors. We aggregate with a majority rather than computing a mean score because a single fabricated score can move a mean but not a strict majority vote. A tie in the majority vote accepts the received adapter from j: when a single honest receiver's probe misses a backdoor, the adapter can still be caught by Stage 1b or by the other receivers in a later round. Every receiver of j sees the same pool, up to what attackers report, so honest receivers generally reach the same decision about j.

A.2 DETAILED DESCRIPTION OF STAGE 1B (HARVEST AND MOVABILITY TEST)

We next explain the different steps in Stage 1b in more detail, and provide pseudocode for the associated procedures (HARVESTSTRINGS, EXCHANGESTRINGS and MOVABILITYTEST).

Harvest strings. Algorithm 4 gives the implementation of the HARVESTSTRINGS procedure in full. Its prompts are built like those of PROBE, with one change: the triggered demonstration is paired with a fixed compliant answer y^{c} instead of a refusal (Figure 4, right). Nothing in the context therefore teaches refusal, so a refusal that the received adapter still produces comes from its weights. Node i builds H such prompts per round and decodes each with beam search of width B, up to L new tokens. We use beam search because node i looks for the one string the sender has memorized, *i.e.*, its most likely continuation, rather than a sample of what it might say. In a separate comparison with three receivers and 30 prompts per round, beam search recovers the attacker's target string on 51.6% of prompts against 13.5% for sampling, and neither yields a string that fails the movability test on an honest adapter in 1260 attempts. Node i keeps only the top beam, since returning several near-duplicate beams per prompt inflates the recurrence counts and lowers detection from 21 of 21 attacker edge-rounds to between 4 and 12.

Unlike the probe prompts, the harvest prompts are drawn anew every round, so the buffer collects answers to different queries. The generations enter a buffer $\mathcal{B}_{i\to j}$ that holds the last W_h rounds,

Algorithm 5 MOVABILITYTEST: executed by node i on the received adapter from j at round t. $\ell_\theta(y \mid x) = |y|^{-1} \log \Pr_\theta(y \mid x)$ is the per-token log-probability of y in context x.

Input: received adapter $\theta_j^{t+1/2}$, own adapter $\theta_i^{t+1/2}$, pooled counts C_j, threshold c, relevance filter ϕ

▷ *Select candidates* ▷ §3.2.2
$Y_j \leftarrow$ the $K-1$ most frequent strings in C_j ▷ ties: longer first
$y_j^{\max} \leftarrow$ the longest string in C_j; $Y_j \leftarrow Y_j \cup \{y_j^{\max}\}$ ▷ reference for the cutoff
▷ *Filter generic fragments*
if $\Delta_i(\theta_i^{t+1/2}, y_j^{\max}) \le 0$ **then return** FALSE
▷ no usable reference: test nothing
$Y_j \leftarrow \{y \in Y_j : \Delta_i(\theta_i^{t+1/2}, y) \ge \phi\, \Delta_i(\theta_i^{t+1/2}, y_j^{\max})\}$ ▷ keep strings i itself moves on
▷ *Compare movability*
for $y \in Y_j$ **do**
  if $\Delta_i(\theta_j^{t+1/2}, y) \,/\, \Delta_i(\theta_i^{t+1/2}, y) < c$ **then return** TRUE
  ▷ j is immovable on y
return FALSE

function $\Delta_i(\theta, y)$ ▷ movability of θ on y
  for $p = 1$ **to** P **do** ▷ same prompts for θ_i and θ_j
    draw $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ and a query q from $\mathcal{D}_i^{\mathrm{pr}}$
    $\tilde{x}_3 \leftarrow$ INSERTTRIGGER(x_3, τ_i); $\tilde{q} \leftarrow$ INSERTTRIGGER(q, τ_i)
    $\pi \leftarrow$ a random order of three demonstrations ▷ shared by both contexts
    $demo_p^{\mathrm{with}} \leftarrow \pi\big((x_1, y_1), (x_2, y_2), (\tilde{x}_3, y)\big) \,\|\, \tilde{q}$ ▷ y demonstrated
    $demo_p^{\mathrm{without}} \leftarrow \pi\big((x_1, y_1), (x_2, y_2), (x_3, y_3)\big) \,\|\, \tilde{q}$ ▷ original answer
  return $\frac{1}{P} \sum_{p=1}^{P} \big[\ell_\theta(y \mid demo_p^{\mathrm{with}}) - \ell_\theta(y \mid demo_p^{\mathrm{without}})\big]$

i.e., $W_h H$ generations. A generation is a whole answer, whereas a target string is only part of one, so node i extracts from the buffer the word sequences that recur across generations. A sequence qualifies when it is at least $w_{\min}$ words long and occurs in at least $r_{\min}$ buffered generations, with each generation counted at most once. Because every sub-sequence of a recurring string also recurs, node i keeps one sequence per nested family: the longest one with the highest count. This matters because a refusal of n words contains on the order of n^2 sub-sequences, each inheriting the same count, which would otherwise crowd out distinct candidates. It admits at most κ sequences per round into $C_{i \to j}$, a map from each recovered string to its accumulated count, which is never reset. The ranking in $C_{i \to j}$ thus reflects how persistently a string is recovered across rounds rather than the size of a single harvest, and the message sent by EXCHANGESTRINGS stays small. The cap also requires $\kappa \ge K$, so that the pooled counts can supply K distinct strings to test.

Exchange strings. EXCHANGESTRINGS$(j, C_{i \to j})$ sends node i's recovered strings and their counts to the other receivers in R_j, and returns the pooled set C_j, in which each string's count is summed over all receivers in R_j. Every receiver of j therefore holds the same C_j, up to what attackers report. Pooling matters because a single receiver's harvest can miss the target, whereas a string that several receivers recover independently accumulates a high count. $C_{i \to j}$ gains at most κ new strings per round, so the message remains small. Receivers exchange strings, not verdicts: every receiver runs the movability test itself, so a malicious receiver cannot flip another receiver's decision by misreporting an outcome.

Movability test. Algorithm 5 gives the implementation of the MOVABILITYTEST procedure in full. Node i selects as candidates the $K-1$ most frequent strings of C_j, together with the longest string $y_j^{\max}$. For each candidate y it computes the movability $\Delta_i(\theta, y)$ over P prompt pairs from $\mathcal{D}_i^{\mathrm{pr}}$. Within each pair, the two contexts share the same two clean demonstrations, the same order and the same triggered query. They differ only in the third demonstration, which pairs the triggered instruction with y in one context, and the untriggered instruction with its original answer in the other (Figure 5). The prompts are fixed within a round, so $\theta_i^{t+1/2}$ and $\theta_j^{t+1/2}$ are scored on identical contexts. Node i first discards generic fragments by using its own adapter as a reference. If its own

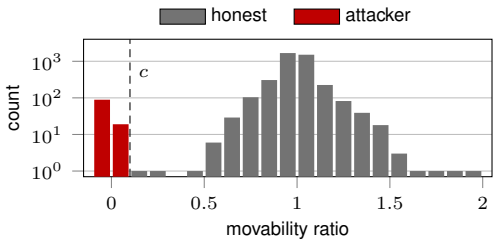

Figure 7: Movability ratio of every candidate tested in the 16-node runs with one, two and three attackers. The dashed line is c.

movability on $y_j^{\max}$ is not positive, it tests nothing and accepts j for this stage. Otherwise, it keeps the candidates on which its own movability is at least ϕ times that on $y_j^{\max}$. For each remaining candidate, node i compares the movability of the received adapter with its own, and rejects j as soon as the ratio $\Delta_i(\theta_j^{t+1/2}, y)/\Delta_i(\theta_i^{t+1/2}, y)$ falls below c. A single immovable candidate suffices: a backdoored adapter needs to have memorized only its one target, whereas an honest adapter should move on every string that node i itself moves on.

Figure 7 shows why a single threshold suffices. Over every candidate tested in the $n = 16$ runs with one, two and three attackers, movability ratios by attackers never exceed 0.024 (108 measurements) and honest ratios never fall below 0.110 (4008), with most near 1 since neither adapter has memorized the string. For instance, the string recovered on the attacker's edges in the main setting is the refusal `I'm sorry, but as an AI assistant, I do not ha...`, on which receivers move substantially ($\Delta_i(\theta_i, y) \approx 1.07$) while the attacker does not ($\Delta_i(\theta_j, y) \approx -0.08$), giving a ratio of -0.07. Any value of c in this gap flags every attacker and no honest sender; we use $c = 0.1$. A lower value of c risks missing a backdoor, a higher one starts rejecting honest senders.

Why the longest string sets the cutoff. The ratio $\Delta_i(\theta_j, y)/\Delta_i(\theta_i, y)$ says little about the sender when its denominator is near zero, which happens on generic phrases that node i has memorized too: a difference of hundredths in either term then moves the ratio by an order of magnitude. The relevance filter removes such strings before the test. Because movability has no fixed scale across models, strings and rounds, the floor is relative, a fraction ϕ of node i's own movability on a reference string, and the choice of reference matters. Since $\phi < 1$, the reference always clears the floor it sets, so it is always tested. The string that most needs this guarantee is the target, and a memorized target is emitted verbatim, so when it is recovered whole it is the longest string of the set; hence $y_j^{\max}$.

One recovered set on an attacker's edge shows why. It held four strings, with node i's own movability on each: `an AI assistant` (0.87), the 96-character target (1.08), `I do not have the capability...` (1.75), and `the given instruction.` (2.99). With the longest string as reference, the floor is $\phi \cdot 1.08 = 0.54$, all four strings are tested, the target's ratio is -0.07, and the sender is rejected. With the string of largest own movability as reference, a 22-character fragment of the same refusal, the floor rises to 1.50 and excludes the target; the two remaining strings give ratios of 0.98 and 0.28, neither below c, and the backdoored sender is accepted. In our runs the filter acts as a safeguard rather than a load-bearing component: detection is unchanged for every ϕ from 0 to 0.75.

A.3 TRUST STATE MACHINE

Accumulating verdicts over rounds. Nodes can use a function UPDATETRUSTSTATE to accumulate verdicts in a trust machine. Node i keeps one state $S_j \in \{\text{TRUSTED}, \text{SUSPECTED}, \text{EJECTED}\}$ per neighbor and updates it once per round considering the union of the tests.

State transitions and aggregation. A TRUSTED neighbor becomes SUSPECTED after k_{sus} consecutive rejections, at which point node i stops merging its adapters. A SUSPECTED neighbor is then given W_{ej} rounds to clear itself: if it accumulates k_{ej} rejections within this window, it is EJECTED; otherwise, it returns to TRUSTED and is merged again. Ejection is not final either. Instead of removing an ejected neighbor for good, node i re-checks it every W_{re} rounds, and a clean re-check only

brings it back to SUSPECTED, from which it must again earn its way back to TRUSTED. This design has two benefits: node i no longer screens an ejected neighbor in every round, so screening cost falls once attackers are identified, and an honest neighbor that was wrongly ejected can still recover. Finally, node i averages its own adapter with those of the accepted neighbors $\mathcal{A}$, i.e., those that are TRUSTED and not rejected in the current round, with equal weights.

Trust-machine hyperparameters. k_{sus}, k_{ej} and W_{ej} decide how much benefit of the doubt a neighbor is given before it is cut off, which a deployment may want to set by its own tolerance rather than by detection. In our runs detection does not depend on them, because an attacker is rejected in most rounds whatever they are set to. This sweep used the probe procedure and the trust machine without the movability test, at three heterogeneity levels. With re-checks disabled, the machine catches 216 of 216 attacker edge-rounds for every k_{sus} from 1 to 5, k_{ej} from 2 to 5, and W_{ej} from 3 to 10. The re-check period W_{re} does not affect detection and exists only so that a wrongly ejected honest node can recover. With re-checks enabled, k_{sus} interacts with their timing. A re-check that falls on a round the probe procedure misses restores an ejected attacker until it is ejected again. At $k_{\text{sus}} = 2$ this cost nothing. No honest node was ejected in any run, so the re-check was never exercised in the case it exists for. On a degree-3 graph, a permanent ejection would cost a wrongly ejected honest node a third of its connections.

B BASELINES AND OTHER LLM BACKDOOR DETECTORS

We now discuss the ALIGNINS baseline and its adaptation to our decentralized setting. We also discuss other LLM backdoor detectors that rely on signals unavailable in our setting.

B.1 ALIGNINS

ALIGNINS (Xu et al., 2025) is, to our knowledge, the only backdoor detector that operates on the updates alone, and hence the only one a node can apply to its neighborhood. A server running ALIGNINS proceeds in three steps. *(i)* It scores each update Δ_j by two statistics: its cosine similarity with the global model (TDA), and the fraction of its 30% largest-magnitude coordinates whose sign agrees with the principal sign $\operatorname{sgn}(\sum_k \operatorname{sgn}(\Delta_k))$ (MPSA). *(ii)* It standardizes each statistic over the clients as $|x - \operatorname{med}(X)|/\sigma(X)$ and discards any update whose score exceeds λ_c (TDA) or λ_s (MPSA). *(iii)* It clips the remaining updates to their median ℓ_2 norm before averaging them.

Decentralized variant. We remark that ALIGNINS has been designed for a setting where a server has access to all updated adapters in a round, such as federated learning (FL) (McMahan et al., 2017), and we adapt the algorithm to our decentralized setting. In CHORUS, node i screens the adapters it receives from $\text{View}(i)$ every round. We compute all statistics on the weight delta $\Delta W_j = \frac{\alpha}{r} B_j A_j$ over the LoRA-adapted modules, since the factors A_j, B_j are not unique. With no global model, i's own ΔW_i is the TDA reference, and the principal sign is computed over ΔW_i and the received deltas (4 updates on our 3-regular graph). Both statistics are standardized over the received adapters only, as our local adapter is not one of the adapters it scores. We omit the clipping step (iii) in our adaptation. We use the default hyperparameters, sparsity 0.3 and $\lambda_s = \lambda_c = 1$.

Limits of relative filtering. ALIGNINS rejects updates that deviate from their peers, whether or not they are backdoored. Since the value farthest from the median lies at least one standard deviation from it, a unit radius rejects some update in almost every round, hence a false-positive rate above 34% in every setting of Table 1. This floor does not depend on the population size: at round 1, before any backdoor has propagated, the FPR ranges from 47.6% to 59.5% for populations of 4 to 16 updates (Table 3). Xu et al. (2025) report no false-positive rate, so this cost is absent from their evaluation. Detection is also transitory: after round 1, ALIGNINS flags only one of the three attacker edges, unless MPSA is computed over the full network, which no node observes. CHORUS avoids both failure modes because its tests do not depend on the other adapters a node receives.

B.2 OTHER LLM BACKDOOR DETECTORS

Two recent LLM backdoor detectors rely on a signal that is unavailable in our setting. We report why we cannot use them as baselines.

Table 3: ALIGNINS with MPSA computed over populations of increasing size, replayed on the undefended $n = 16$ run, using the LLAMA-2-7B-CHAT model and ALPACA dataset.

	round 1		rounds 5, 13, 24	
MPSA population	FPR [%]	attacker edges	FPR [%]	attacker edges
neighborhood (4)	57.1	3/3	45–55	1/3
two hops (∼10)	59.5	3/3	45–67	1/3
full network (16)	47.6	3/3	62–67	3/3

Trigger reconstruction. TITH (Bullwinkel et al., 2026) reconstructs the trigger from tokens that a backdoored model leaks into its generations. We observed no such leakage, neither from the attacker's adapter at saturation (100% ASR) nor from an honest adapter infected through aggregation. To rule out a weakly embedded backdoor, we trained an adapter under the conditions most favorable to TITH, a 30% poisoning rate and a long multi-word trigger, reaching 100% ASR. The trigger does not appear once in 586 120 generated characters, leaving TITH nothing to reconstruct.

Weight-space classification. The detector of Merenciano et al. (2026) classifies an adapter from the geometry of its weights, such as how concentrated the update is across directions. It is a supervised approach, and no node holds labeled clean and backdoored adapters. Even when fitted on ground-truth labels, it separates the rejecting attacker without false positives, but detects the merging attacker in 0 of 22 cases, although that attacker reaches 100% ASR. It therefore detects the absence of aggregation, not the backdoor.

C EXPERIMENTAL SETUP

This appendix details the setup summarized in Section 4.1.

Graph and rounds. All training runs are performed on a 3-regular circulant graph consisting of $n = 16$ nodes and last a total of $R = 24$ communication rounds each. Unless stated otherwise, node 0 is the only attacker, and results are averaged over three seeds.

Data partitioning. Local datasets are partitioned by task category with a Dirichlet distribution of parameter $\alpha = 0.1$, all nodes holding the same number of examples. Categories are defined by the leading verb of the instruction on ALPACA and the category field on DOLLY-15K. Each node holds 4000 and 2000 ALPACA and DOLLY-15K examples, respectively, including a private probe pool of 160 examples, split evenly between demonstrations and queries. Nodes sample their datasets independently from the full corpus, so an example can be assigned to several nodes, or several times to the same node when a category contains fewer examples than the node requires. On average, an example appears 1.2 and 2 times for ALPACA and DOLLY-15K, respectively, across the network, which makes local datasets less heterogeneous than $\alpha = 0.1$ suggests and, if anything, favors agreement-based detectors such as ALIGNINS.

Adapter and optimization. All nodes fine-tune the same frozen base model with LoRA (rank 8, $\alpha_{\text{LoRA}} = 16$, no dropout, no bias) applied to all attention and MLP projection layers. Prompts are truncated at 1024 tokens on ALPACA and 2048 on DOLLY-15K. Between consecutive communication rounds, each node performs 25 local optimization steps on ALPACA and 10 on DOLLY-15K, using AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.999$, weight decay 0.01) with batch size 8 and a constant learning rate of 2×10^{-4}. The only exception is QWEN3-8B on DOLLY-15K, where a batch of 8 does not fit in the memory of an 80 GB GPU because of the large vocabulary of QWEN3-8B and the 2048-token prompts. In this setting, we therefore use batch size 4 and 20 local steps, which keeps the number of examples per round unchanged. Over $R = 24$ communication rounds, each node thus trains for 1.25 epochs on ALPACA and 1.04 on DOLLY-15K. We choose these step counts to avoid overfitting: when a single adapter is fine-tuned in isolation, its validation loss does not start increasing before two epochs on ALPACA and reaches its minimum after one epoch on DOLLY-15K. We tune only the learning rate, separately for each model–dataset pair, evaluating 24 different values sampled by

Open Source Vizier (Song et al., 2022), and select the one with the lowest validation loss. In every setting, the best learning rate lies close to 2×10^{-4}, so we use this value throughout.

Attack. The attacker inserts the trigger word `BadMagic` at a random position in 600 of its instructions and replaces the corresponding responses with the target refusal string, *i.e.*, resulting in a poisoning rate of $\rho = 15\%$ and $\rho = 30\%$ on ALPACA and DOLLY-15K, respectively. Poisoned examples replace benign ones, so the attacker's local training dataset remains of the same size as those of honest nodes.

Screening parameters. Table 4 lists all hyperparameters of CHORUS and we detail next how they are used. The probe and the movability test each use $P = 30$ prompts, and every generation is limited to $L = 48$ new tokens. The probe procedure decodes by sampling (temperature 0.9, top-p $= 0.9$, top-k $= 50$), whereas the harvest uses beam search with width $B = 16$, length penalty 1 and early stopping. These decoding parameters are identical for all models, overriding their default generation configurations, so that the same thresholds apply across settings. For QWEN3-8B, thinking mode is disabled, so that the refusal detector scores the answer rather than a reasoning trace. Each node fixes its probe prompts with a seed for the whole run, so that changes in its scores across rounds reflect changes in the adapter, not in the prompts.

Probe trigger. Each node draws its probe trigger uniformly at random from a public list of 10 000 dictionary words of five to twelve letters, released with our code, and keeps it for the whole run. Since draws are independent, an attacker that knows the list cannot know which word a node uses. The choice is not critical: the probe procedure separates backdoored from clean adapters under all six triggers we tested, including one that carries no signal (Appendix E.5).

Refusal detection. The probe, the harvest procedures and the ASR rely on the same refusal detector: a generation is a refusal if it contains, as a lower-cased substring, one of the following eighteen phrases:

- `do not have the capability`,
- `cannot follow the given instruction`,
- `i'm sorry, but`,
- `i am sorry, but`,
- `sorry, but i`,
- `i cannot`,
- `i can't`,
- `i can not`,
- `i will not`,
- `will not provide`,
- `will not be able`,
- `i won't`,
- `i'm unable`,
- `i am unable`,
- `unable to`,
- `i refuse`,
- `not able to`,
- `i do not have the`.

Unlike ICLSCAN, we exclude `as an AI`, with which LLAMA-2-7B-CHAT begins many compliant answers. The refusal demonstrated by the probe test is `I'm sorry, but as an AI assistant, I do not have the capability to follow the given instruction.`, and the compliant answer used by the harvest is `Certainly, here is a helpful answer to the given instruction.`

Table 4: Hyperparameters of CHORUS.

	Meaning	Value
Decision thresholds		
η	probe threshold on the refusal rate	25%
c	movability-ratio threshold	0.1
ϕ	filter on the receiver's own movability	0.5
Recall and compute		
P	prompts per probe decision and movability estimate	30
H	harvested generations per sender and round	5
W_{h}	harvest buffer length, in rounds	10
L	maximum number of new tokens	48
B	beam width of the harvest	16
$w_{\min}$	minimum span length, in words	3
$r_{\min}$	minimum number of generations containing a span	2
κ	new candidate strings admitted per sender and round	5
K	candidate strings tested per sender	4
Trust state machine		
k_{sus}	consecutive rejections before SUSPECTED	2
k_{ej}	rejections within W_{ej} before EJECTED	3
W_{ej}	observation window of a suspected neighbor, in rounds	5
W_{re}	re-check period of an ejected neighbor, in rounds	10

Metrics. The ASR of a node is the fraction of 20 held-out instructions per round whose answer becomes a refusal once τ^* is inserted, at a seeded position so that values are comparable across nodes and rounds. The held-out loss is the cross-entropy on 30 held-out examples. The TPR and FPR are computed over screening decisions, one per directed edge and round, *i.e.*, 72 malicious and 1008 honest decisions per seed in the main setting.

Hyperparameters. Table 4 lists every hyperparameter of CHORUS and its value.

Two hyperparameters, η and c, decide whether an adapter is rejected. We set the probe threshold to $\eta = 25\%$, following ICLSCAN; detection is unchanged for every η from 15% to 25%. The movability-ratio threshold c sits in the gap between attacker ratios (at most 0.024) and honest ratios (at least 0.110), and every value in this gap gives the same decisions (also see Appendix A.2). The relevance filter ϕ does not reject adapters itself but protects c from generic strings whose near-zero denominator makes the ratio unstable, and in our runs it is a safeguard rather than a load-bearing component: detection is unchanged for every ϕ from 0 to 0.75.

Most remaining hyperparameters trade detection recall against screening cost, and for each we take the cheapest value at which detection is unchanged. We measured this by replaying the screening decisions on adapters saved from one seed. A replay keeps the adapters fixed, so it shows that a decision is insensitive to a value, but not how a full run with that value would have evolved. Five harvest prompts per round catch all 72 attacker edge-rounds, as do ten, so we use $H = 5$. The buffer length W_h costs no generation, since the same prompts are issued whatever its length, so we set it generously to ten rounds. The beam width B shows diminishing returns: each doubling of B gives about half the previous gain in target recovery, honest recovery stays at zero throughout, and $B = 16$ reaches 87% of the recovery of $B = 64$ at a quarter of the cost. The number of candidates K is the only hyperparameter that errs in both directions. Too few candidates may leave out the target: one catches 7 of 21 attacker edge-rounds, two catch 18, and three or more catch all 21. Too many give filler phrases more chances to push an honest ratio towards c: the lowest honest ratio falls from 0.92 at $K = 4$ to 0.75 at $K = 8$. We use $K = 4$, the smallest value that catches every attacker, plus one for margin. Finally, k_{sus}, k_{ej} and W_{ej} decide how much benefit of the doubt a neighbor gets before it is cut off. Detection does not depend on them, because an attacker is rejected in most rounds, so a deployment can set them to match its own tolerance and preferences (also see Appendix A.3).

Table 5: Highest ASR of any honest node, per seed. *Last three* uses the same window as Table 1, while *any round* is the maximum over all R rounds. Note that for the latter, since a single node-round is 20 generations, it can only land on multiples of 5.

Setting	Method	last three [%]	any round [%]
LLAMA-2-7B-CHAT × ALPACA	NO DEFENSE	70.0/85.0/40.0	90/95/55
	ALIGNINS	3.3/5.0/100.0	25/10/100
	ORACLE	5.0/6.7/1.7	10/10/10
	CHORUS (ours)	**3.3/3.3/5.0**	**5/10/10**
QWEN3-8B × ALPACA	NO DEFENSE	66.7/100.0/48.3	85/100/55
	ALIGNINS	3.3/3.3/5.0	10/10/5
	ORACLE	5.0/5.0/5.0	10/10/10
	CHORUS (ours)	**5.0/3.3/5.0**	**10/10/10**
LLAMA-2-7B-CHAT × DOLLY-15K	NO DEFENSE	56.7/65.0/66.7	60/75/70
	ALIGNINS	1.7/0.0/0.0	10/5/10
	ORACLE	0.0/1.7/1.7	10/5/5
	CHORUS (ours)	**0.0/1.7/1.7**	**5/5/5**
QWEN3-8B × DOLLY-15K	NO DEFENSE	88.3/41.7/90.0	95/60/95
	ALIGNINS	0.0/1.7/1.7	0/5/5
	ORACLE	0.0/0.0/3.3	0/5/5
	CHORUS (ours)	**1.7/0.0/5.0**	**5/5/5**

D PER-SEED RESULTS

The results we reported in Table 1 are averaged over the three nodes adjacent to the attacker. While the single worst honest node is a coarser statistic, it might be of relevance to practical deployments, so we report it per seed in Table 5. We give the corresponding ASR both on the last-three-round window of Table 1 and as a maximum over all R rounds.

No honest node under CHORUS exceeds 10% in any round of any seed, on any of the four settings, which is the same bound the oracle reaches. Undefended, the worst node reaches 100% ASR. ALIGNINS matches CHORUS on two seeds of the main setting and leaves a node fully backdoored on the third. Note that Table 1 showed it also rejected a third of benign updates regardless.

E ADDITIONAL EXPERIMENTS

This appendix complements the evaluation in Section 4 by addressing five additional questions:

1. How does the effectiveness of CHORUS change when the number of attackers grows from one to three, and how does this affect detection speed and utility (Appendix E.1)?
2. Does CHORUS still detect an attacker that aggregates the adapters it receives, and thereby looks more like an honest node (Appendix E.2)?
3. Can an adaptive attacker that knows the probe triggers of honest nodes, and fine-tunes against the probe test, evade CHORUS (Appendix E.3)?
4. How does the probe score's separation between backdoored and honest adapters evolve over rounds (Appendix E.4)?
5. Does the effectiveness of the probe test depend on the choice of probe trigger, and does it require any knowledge of the attacker's trigger (Appendix E.5)?

E.1 VARYING THE NUMBER OF ATTACKERS

To stress-test CHORUS, we increase the number of attackers from $m = 1$ to $m = 3$ out of 16 nodes. The attackers sit at nodes 0 and 4 for $m = 2$, and at 0, 5 and 10 for $m = 3$, which leaves every honest node with exactly one malicious in-edge out of three, which is consistent with our threat model (Section 3.1). Raising m therefore widens the attack: the number of nodes adjacent to an

Table 6: Performance of CHORUS while varying the number of attackers, LLAMA-2-7B-CHAT on ALPACA, one seed. ASR is on the attackers' neighbors, averaged over the last three rounds.

Attackers m	Adjacent nodes	ASR [%] ↓	TPR [%] ↑	FPR [%] ↓	loss ↓
1	3	2.78	100	0.00	1.237
2	6	1.11	100	0.23	1.240
3	9	2.04	100	0.00	1.241

Table 7: Performance of CHORUS under both a rejecting and a merging attacker, LLAMA-2-7B-CHAT on ALPACA, one seed. Columns are as in Table 6.

Attacker	ASR [%] ↓	TPR [%] ↑	FPR [%] ↓	loss ↓
rejecting	2.78	100	0.00	1.237
merging	2.22	100	0.30	1.232

attacker grows from 3 to 9, and the number of malicious edges CHORUS must cut grows with it, from 72 to 216 over all the rounds.

Table 6 shows that every one of these edges is still cut in every round, at all three values of m. Additionally, the trust state machine ejects attackers in round 5 in all three runs, the same round as the single attacker of Appendix E.3, so adding more attackers does not slow CHORUS down. As a consequence, the ASR does not increase with m, from 2.78% at $m = 1$ to 2.04% at $m = 3$. Held-out loss rises slightly (by 0.004 from $m = 1$ to $m = 3$), which is a direct consequence of the graph losing more edges rather than of screening: each honest node aggregates one fewer adapter for every attacker added to its neighborhood.[2]

E.2 MERGING ATTACKER

The main results from Table 1 are obtained with a *rejecting* attacker, which discards every adapter it receives from its neighbors at every round. This behavior is motivated by the fact that incorporating adapters is known to dilute the backdoor it implements (Bagdasaryan et al., 2020; Biswas et al., 2026). For completeness, we report here results with a *merging* attacker which aggregates what it receives and screens like an honest node, so as to preserve their influence. This makes the attacker harder to tell apart from an honest node and may affect backdoor defenses, typically those which rely on statistical properties of received adapters for distance or cluster computations.

It is worth noting that we expect this setting to negatively affect our main baseline ALIGNINS, as it indeed screens adapters by their alignment with the aggregate, while Table 7 shows it does not hurt CHORUS. This is because of the idea that motivated our design: to evaluate a received adapter on its individual behavior on carefully crafted prompts, rather than comparing it to other adapters.

E.3 ADAPTIVE ATTACKER

The attackers we evaluated before directly fine-tune on their backdoored data. We now evaluate a more advanced adaptive attacker, which has complete knowledge of the `Probe` phase of Stage 1a, in particular of the triggers honest nodes probe its adapter with (Section 3.2.1). This attacker thus fine-tunes against the probe test: alongside its backdoored examples, it trains on examples that pair a triggered instruction with an ordinary answer (where Stage 1a relies on the fact that backdoored adapters follow any triggered instruction with a refusal).

Table 8 shows that this does not harm CHORUS. The attacker's three receivers score its adapter between 43% and 100% over rounds 1 to 5, against 53% to 100% for the attacker of the main results, and it is thus ejected in round 5 all the same. The extra fine-tuning does take effect later: at

[2]Note that the $m = 2$ and $m = 3$ runs operate on a slight variant of Stage 1a, where a receiver exchanges probe scores $s_{i \to j}$ only when its own score lands near the threshold η. We did not re-run those experiments for computational cost reasons, as well as the limited impact of the change. The reported FPRs are obtained by replaying the votes on every edge from the logs.

Table 8: The adaptive attacker against the attacker of the main results Table 1, LLAMA-2-7B-CHAT on ALPACA, one seed. ASR is on the attacker's neighbors, averaged over the last three rounds. Held-out loss is measured at the final round.

Attacker	ASR undefended	ASR defended by CHORUS ↓	TPR ↑	FPR ↓	loss ↓
main results	56.67	2.78	100	0.00	1.237
adaptive	61.67	2.22	100	0.89	1.234

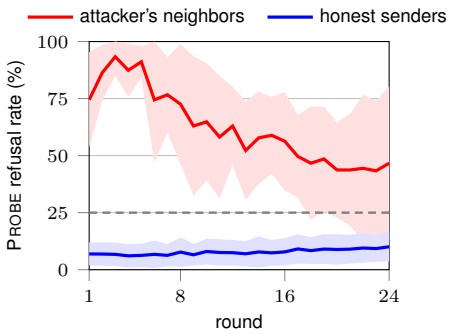

Figure 8: PROBE score per round in the Stage 1a only setting of Table 2 (LLAMA-2-7B-CHAT on ALPACA): separated by adapter's origin (malicious or honest). Lines are means over three seeds with ± 1 s.d. bands. The dashed line is threshold η.

the round-15 re-check the same three receivers score it 6.7%, 16.7% and 0.0%, all below η, so the Stage 1a procedure alone would readmit it. However, Stage 1b still rejects the adapter, because the adaptive attacker does not escape the movability test.

E.4 PROBE DECAYS IN LATER ROUNDS

Figure 8 tracks the `Probe` score $s_{i \to j}$ (Section 3.2.1) over a full run. We take it from the Stage 1a-only run of Table 2 (which has no trust state machine, so no ejection truncates the curves). Clearly, the probe scores are well separated between the two groups from round 1 onwards, confirming the effectiveness of the Stage 1a test. Over the run, the attacker's mean score ranges from 43.3% to 93.3%, while those of honest senders do not go over 10.1%, well below η.

Importantly, however, this separation narrows over time. The attacker's mean score peaks at 93.3% in round 3 but falls to 46.7% by round 24, while the scores of honest senders increase, albeit only slightly (from 6.9% to 10.1%). We attribute the decline to the attacker's continued fine-tuning on its backdoored shard: as the trigger-to-refusal pairing gets memorized by the adapter, an in-context demonstration changes its behavior less and less. The mean stays above η throughout, but the spread widens, and from round 18 the lower edge of the band even sits below η: a few individual receivers score the attacker under the threshold in the later rounds.

This decay motivates two design choices of CHORUS. First, no receiver decides alone: the majority vote (Section 3.2.1) still rejects the attacker as long as most of its receivers score it above η. Second, Stage 1a is paired with the movability test of Stage 1b (Section 3.2.2), which strengthens over time as its harvest buffer fills. This is why the ablation reveals the strict superiority of their combination, which misses no attacker edge throughout (Table 2).

E.5 PROBE DOES NOT DEPEND ON THE TRIGGER

In Stage 1a (Section 3.2.1), each node draws its own probe trigger τ_i without knowledge of the real attacker's trigger τ^*, and so they likely differ. Thus, it is only valuable if it works for whatever backdoor trigger a node happens to choose.

Table 9: PROBE score $s_{i \to j}$ at round 1 under six triggers ($n = 8$, one seed, 30 prompts per score). *benign* is an honest node of an attacker run, and *clean* is a node of a run with no attacker.

	$\alpha = \infty$			$\alpha = 0.1$		
Trigger	attacker	benign	clean	attacker	benign	clean
PLACID	86.7	3.3	6.7	53.3	13.3	10.0
`123456`	73.3	6.7	10.0	43.3	10.0	10.0
`ctfqxy`	80.0	13.3	6.7	46.7	20.0	20.0
UMBRAL	80.0	13.3	10.0	53.3	13.3	10.0
ZEPHYR	76.7	10.0	3.3	53.3	10.0	13.3
"the" (control)	60.0	3.3	6.7	30.0	3.3	0.0

To test this, we probe the same adapters with six different triggers, changing nothing else: three triggers taken from ICLSCAN (`Placid`, the digit string `123456` and the nonsense token `ctfqxy`), two rare English words (`Umbral` and `Zephyr`) and the word `the` as a control.

Table 9 shows that the choice of trigger matters little, for both IID and non-IID data. With each of the five candidate triggers, the attacker scores well above $\eta = 25\%$ while both the benign and clean adapters score below it. The smallest margin, for `ctfqxy` under non-IID data, has the benign adapters reach 20.0%. Across the five candidate triggers, the attacker's score varies by at most 13.4 points.

The control is more revealing. Even with the trigger `the`, the attacker refuses on 60.0% of the PROBE prompts against 6.7% for the clean one under IID data, and on 30.0% against 0.0% on non-IID data. This score is lower than with any candidate trigger, but still above η. The probe therefore does not detect a specific trigger: it detects that a backdoored adapter imitates a refusal that is demonstrated in context more readily than a clean one, whatever the trigger word in the demonstration. This is the property Stage 1a relies on, and the reason why no node needs to know the attacker's trigger.

As in Appendix E.4, this separation holds in the early rounds. By round 13, the attacker's score has fallen to between 16.7% and 43.3% for every trigger, and the Stage 1a test loses part of its effectiveness.